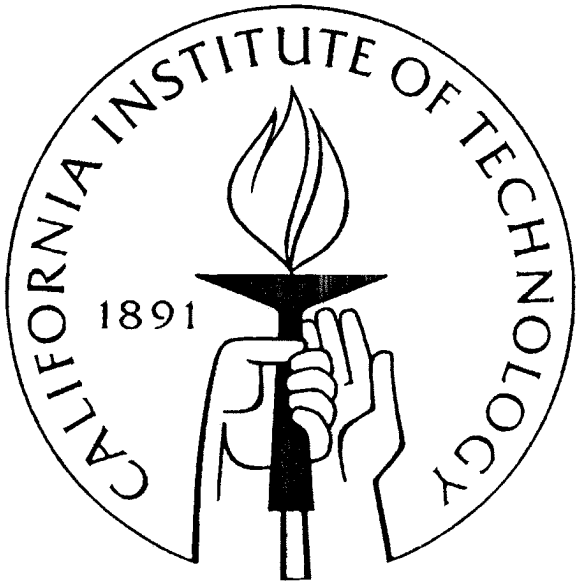

A Parabolic Theory of Load Balance

Alan Heirich
Stephen Taylor

Computer Science Department
California Institute of Technology

Caltech-CS-TR-93-25

A Parabolic Theory of Load Balance [1]

Alan Heirich & Stephen Taylor
Scalable Concurrent Programming Laboratory
California Institute of Technology

Abstract

We derive analytical results for a dynamic load balancing algorithm modeled by the heat equation $u_t = \nabla^2 u$. The model is appropriate for quickly diffusing disturbances in a local region of a computational domain without affecting other parts of the domain. The algorithm is useful for problems in computational fluid dynamics which involve moving boundaries and adaptive grids implemented on mesh-connected multicomputers. The algorithm preserves task locality and uses only local communication. Resulting load distributions approximate time asymptotic solutions of the heat equation. As a consequence it is possible to predict both the rate of convergence and the quality of the final load distribution. These predictions suggest that a typical imbalance on a multicomputer with over a million processors can be reduced by one order of magnitude after 105 arithmetic operations at each processor. For large n the time complexity to reduce the expected imbalance is effectively independent of n.

[1]The research described in this report is sponsored primarily by the Defense Advanced Research Projects Agency, DARPA Order number 8176, and monitored by the Office of Naval Research under contract number N00014-91-J-1986.

1. INTRODUCTION

The emergence of massively parallel computers has introduced new considerations for designers of concurrent algorithms. Two primary issues when dealing with irregular problems are *load balancing* and *scalability.* A variety of algorithms for dynamic load balancing exist for systems with small numbers of computers [2]. These algorithms are elegant, efficient and provably correct. Unfortunately they require that each processor communicate information about it's workload to a central processor which performs the computation. One of our research goals is to develop algorithms which scale to the next generation of fine-grain mesh-connected multicomputers [10, 11] which have potentially hundreds of thousands of processors. Therefore we seek algorithms which emphasize nearest neighbor communication and distribute the computational work across all of the affected processors.

We frequently encounter the need for dynamic load balancing in our work on computational fluid dynamics. This occurs in finite volume or finite difference simulations in which we have partitioned the computational domain across a set of processors on a multicomputer. We can initially distribute the computational grid points in such a way as to make the load on each processor approximately the same. The load remains balanced until the computational grid is adapted to follow shock waves or moving boundaries. These adaptations create grid points in some parts of the domain and destroy points in other parts. As a result the loads on the processors become unequal and dynamic rebalancing is necessary.

This paper describes an efficient algorithm to diffuse local load imbalances on a mesh-connected multicomputer. The algorithm is parameterized and can achieve any desired refinement of balance to the limits of machine precision. It uses no global communication although it does require that processors at exterior points communicate with their counterparts on the opposite sides of the mesh. (We are presently considering strategies to eliminate this requirement.) The algorithm can be used to rebalance a local portion of a computational domain while the normal computation executes concurrently over the rest of the domain. The algorithm can be implemented using policies to minimize the distances over which tasks migrate during their lifetime. This is a desirable property for many simulation problems because it minimizes the cost of communication during the simulation.

We have chosen to model the process of rebalancing by the heat equation

$u_t = \nabla^2 u$ on a domain with periodic boundary conditions. Heat diffusion is an apt analogy for the types of problems that we encounter. A load imbalance is typically caused by a computational grid adapting locally and asynchronously. This is analogous to creating a heat source or sink at a point inside a physical volume. Just as heat diffuses away from the source or toward the sink in the physical volume we want work to diffuse away from overloaded processors or toward underutilized processors in the multicomputer. The heat equation is an efficient numerical model for diffusing point sources as it exhibits exponential convergence to equilibrium of all Fourier modes and fast exponential convergence of high wavenumber modes.

Similar informal notions have appeared in the literature [3] such as the Rediflow algorithm of Keller et al [8]. Although such approaches are intuitively persuasive they may also be incorrect and lead to unbalanced loads or unstable behavior. Our algorithm is the first "diffusive" method for mesh architectures which is scalable and has rigorous proofs of correctness and convergence. It is also the first effort based on a formal model of the rebalancing process and the first demonstration that established numerical methods for grid based computations can lead to practical algorithms for mesh-connected multicomputers. We are currently exploring implementations of this and other models.

The first analytical work on diffusive dynamic load balancing is due to Cybenko [4]. In addition to an optimal diffusive method for hypercube architectures he presents an elegant analysis of a general iterative scheme for arbitrary interconnection patterns. His scheme distributes the computational work across all of the affected processors but does not restrict itself to nearest neighbor communication. Hong, Tan and Chen [6] appreciate the importance of local communication and derive convergence results for nearest neighbor schemes on hypercubes. Unfortunately their results do not apply to mesh-connected architectures and do not take advantage of numerical solution methods for differential equations.

2. THE DIFFUSIVE HEAT BALANCING ALGORITHM

We interpret the workload as a vector $\vec{u}$ which is distributed across the processors of a three dimensional mesh-connected multicomputer of n processors. The algorithm simulates the passage of "artificial time" during which the workload diffuses according to $u_t = \nabla^2 u$. After this time has elapsed the maximum imbalance $|u_{x,y,z} - \bar{u}|$ has been reduced by a given factor α.

1. **Initialization:**

   Choose a desired reduction α in load imbalance. Solve the following inequality numerically for τ (see table I for example solutions)

$$\frac{8}{n}\sum_{i,j,k}\left[1+\alpha 2\left(3-\cos\frac{2\pi i}{n^{1/3}}-\cos\frac{2\pi j}{n^{1/3}}-\cos\frac{2\pi k}{n^{1/3}}\right)\right]^{-\tau}\leq\alpha \quad (1)$$

   where i,j,k are indexed from 0 to $\left(n^{1/3}\right)/2-1$ and the case $i=j=k=0$ is omitted. Determine a parameter ν related to accuracy of the resulting solution.

$$\nu=\left\lceil\frac{\ln(\alpha)}{\ln\left(\frac{6\alpha}{1+6\alpha}\right)}\right\rceil \quad (2)$$

2. **Rebalancing:**

   At every processor x,y,z adjust the workload $u_{x,y,z}$

   `for l=1 to` $\lceil\tau\rceil$ /* outer loop */

   $u_0=u_{x,y,z}$

   `for m=1 to` ν /* inner loop */

$$u^{(m)}_{x,y,z}=\frac{u_0}{1+6\alpha}+\left(\frac{\alpha}{1+6\alpha}\right)\left(u^{(m-1)}_{x+1,y,z}+u^{(m-1)}_{x-1,y,z}+u^{(m-1)}_{x,y+1,z}+u^{(m-1)}_{x,y-1,z}+u^{(m-1)}_{x,y,z+1}+u^{(m-1)}_{x,y,z-1}\right) \quad (3)$$

   `endfor` /* inner loop */

   Exchange $\left(\alpha u^{(\nu)}_{x,y,z}-\alpha u'^{(\nu)}\right)$ units of work with every neighbor u'.

   $u_{x,y,z}=u^{(\nu)}_{x,y,z}$

   `endfor` /* outer loop */

DISCUSSION

The algorithm consists of two loops. We show in the following sections of this paper that these loops compute an approximate solution to a diffusive process

$\tau(\alpha, n)$		n (total processors)				
		512	4,096	32K	256K	10^6
	0.1	8	6	6	5	5
α	0.01	297	302	245	214	204
	0.001	6,605	12,589	13,795	11,044	9,895

Table 1: outer loop iterations as a function of multicomputer size n and quality of balance α.

described by the heat equation $u_t = \nabla^2 u$. The outer loop is responsible for advancing artificial time by a step dt. This proceeds until a time has elapsed that is sufficient to diffuse the load imbalance. The inner loop is responsible for computing the new value of the load at a specific time step $t = (l)dt$. When the inner loop has computed the new value an exchange operation causes the load at each processor to equal this value.

Each processor communicates only with it's six neighbors in the three dimensional mesh. A processor which is at an exterior mesh boundary may lack as many as three of these neighbors in the physical mesh. These exterior processors communicate with "logical" neighbors at exterior points on opposite sides of the mesh. This makes the mesh logically spherical and implements the periodic boundary conditions on which our analysis depends.

The cost of the rebalancing phase is the sum of two costs: the cost to compute the new load, represented by the inner loop, and the cost to exchange work among the processors. Each inner loop iteration (3) requires seven arithmetic operations and six bidirectional exchanges of a single number at each processor. The total number of iterations is the product $\tau\nu$. The value of ν is always ≤ 3 for $0 < \alpha < 1$. As table I shows τ is 5 for a rebalancing operation involving one million processors when α is 0.1. Assuming a communication rate within a factor of ten of the instruction rate the cost to reduce the expected load imbalance on a million processor multicomputer by a factor $\alpha = 0.1$ is about $10\nu(\alpha)\tau(\alpha, n) = 10(3)5 = 150$ instruction cycles. This requires less than 4 μs of real time on a multicomputer with 40 MHz processors [11].

For fluid dynamics simulations we would like to preserve task locality while we redistribute portions of the computational domain. This is true for many simulation problems because communication rates are highest between adjacent portions of the domain and as a consequence communication cost

is inversely related to locality. In this algorithm locality can be preserved by using an appropriate exchange policy. One such policy would be to assume units of work represent portions of a domain which has been decomposed contiguously across the processors. Under this assumption each neighboring processor represents a neighboring portion of the domain. Any exchange policy which minimizes average pairwise distance between units of work on a common processor maximizes locality.

We would also like to rebalance local portions of a domain without having to interrupt the rest of the computation. The algorithm can be implemented in this way simply by restricting it to a rectangular subportion of the multicomputer mesh. If the algorithm were restricted to a cube then in the following analysis the term n would represent the number of processors in the cube. If the restricted subportion were not a cube then n would be taken as the length of the longest side raised to the third power. This would ensure that convergence bounds reflect worst case estimates.

The reader should understand that the parameter α in the following analysis represents a *reduction* in the expected imbalance rather than a *measure* of the final imbalance. Actual measurements of imbalance require global communication which is contrary to our stated goals. Various implementation policies can be formulated to achieve guaranteed measures of imbalance. For example if the size of the initial disturbance is known then α can be set to remove this disturbance. In this paper we avoid further discussion of the numerous options which exist for implementation policies. Our goal is to establish a sound theoretical basis on which to implement practical algorithms. We are presently considering implementation policies so that we can incorporate these results into our ongoing work in computational fluid dynamics.

3. DERIVATION OF THE REBALANCING PHASE

In this section we demonstrate the relationship between our algorithm and the process of diffusion described by the heat equation $u_t = \nabla^2 u$. We first derive a finite difference approximation to the heat equation. This representation is implicit which means that in order to advance the process of diffusion it is necessary to invert a coefficient matrix. We choose to invert the matrix by Jacobi iteration and thus we arrive at the inner loop (3) of the algorithm.

THE HEAT EQUATION AND FINITE DIFFERENCES

Consider the parabolic heat equation in three dimensions

$$u_t = \nabla^2 u = u_{xx} + u_{yy} + u_{zz} \tag{4}$$

Taylor expanding in t with all derivatives evaluated at (x, y, z, t)

$$\begin{aligned} u(x,y,z,t+dt) &= u(x,y,z,t) + u_t dt + \mathrm{O}(dt^2) \\ u_t &= \left(\frac{u(x,y,z,t+dt) - u(x,y,z,t)}{dt} \right) + \mathrm{O}(dt) \end{aligned}$$

We obtain the second order terms by expanding in spatial variables where omitted coordinates are interpreted as $(x, y, z, t + dt)$

$$\begin{aligned} u(x+dx,\cdot,\cdot,\cdot) &= u(x,\cdot,\cdot,\cdot) + u_x dx + \\ &\quad u_{xx}\frac{dx^2}{2} + u_{xxx}\frac{dx^3}{6} + \mathrm{O}(dx^4) \\ u(x-dx,\cdot,\cdot,\cdot) &= u(x,\cdot,\cdot,\cdot) - u_x dx + u_{xx}\frac{dx^2}{2} - \\ &\quad u_{xxx}\frac{dx^3}{6} + \mathrm{O}(dx^4) \\ u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) &= 2u(x,\cdot,\cdot,\cdot) + u_{xx}dx^2 + \mathrm{O}(dx^4) \\ u_{xx} = \left(\frac{u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) - 2u(x,\cdot,\cdot,\cdot)}{dx^2} \right) &+ \mathrm{O}(dx^2) \end{aligned}$$

Similar expansions in y, z show that the heat equation can be rewritten

$$\begin{aligned} \frac{u(\cdot,\cdot,\cdot,t+dt) - u(\cdot,\cdot,\cdot,t)}{dt} &= \left(\frac{u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) - 2u(\cdot,\cdot,\cdot,\cdot)}{dx^2} \right) \\ &+ \left(\frac{u(\cdot,y+dy,\cdot,\cdot) + u(\cdot,y-dy,\cdot,\cdot) - 2u(\cdot,\cdot,\cdot,\cdot)}{dy^2} \right) \\ + \left(\frac{u(\cdot,\cdot,z+dz,\cdot) + u(\cdot,\cdot,z-dz,\cdot) - 2u(\cdot,\cdot,\cdot,\cdot)}{dz^2} \right) &+ \mathrm{O}(dt, dx^2, dy^2, dz^2) \end{aligned}$$

Taking $dx = dy = dz$ we can express $u(x, y, z, t)$ in terms of time $t + dt$ and approximate the heat equation to within $\mathrm{O}(dt, dx^2)$

$$\begin{aligned} u(x,y,z,t) \approx \left(1 + 6\tfrac{dt}{dx^2}\right) u(\cdot,\cdot,\cdot,t+dt) - \tfrac{dt}{dx^2} [u(x+dx,\cdot,\cdot,\cdot) + u(x-dx,\cdot,\cdot,\cdot) \\ + u(\cdot,y+dy,\cdot,\cdot) + u(\cdot,y-dy,\cdot,\cdot) + u(\cdot,\cdot,z+dz,\cdot) + u(\cdot,\cdot,z-dz,\cdot)] \end{aligned}$$

Notice that if we let $\alpha = \frac{dt}{dx^2}$ the preceding equation can be rewritten

$$u_{x,y,z}(t) = (1+6\alpha)\,u_{x,y,z}(t+dt) - \alpha\,[u_{x+1,y,z}(t+dt) + u_{x-1,y,z}(t+dt) \\ +u_{x,y+1,z}(t+dt) + u_{x,y-1,z}(t+dt) + u_{x,y,z+1}(t+dt) + u_{x,y,z-1}(t+dt)] \tag{5}$$

SIMULATING THE PROCESS OF DIFFUSION

We can consider (5) as a vector equation if we assign coordinates x, y, z to the elements of a vector $\vec{u}$ in the natural way,

$$\vec{u}(t) = A\vec{u}(t+dt) \tag{6}$$

All terms in $\vec{u}$ on the right hand side represent values at time $t+dt$. A finite difference scheme of this sort is said to be "implicit" because values at time t are expressed as a function of values at time $t+dt$. Implicit schemes are known to have desirable stability properties. In particular the error terms do not accumulate in successive solutions so the error in the final solution is $O(\alpha)$. In order to compute solutions at successive time intervals dt we must invert the coefficient matrix A to solve

$$\vec{u}(t+dt) = A^{-1}\vec{u}(t) \tag{7}$$

There are many possible ways to compute $A^{-1}\vec{u}(t)$. In this paper we consider the method of Jacobi iteration [5] as it maintains locality of communication and therefore provides a scalable algorithm. This method determines $\vec{u}(t+dt)$ in solving the system $A\vec{u}(t+dt) = \vec{u}(t)$ with $\vec{u}(t)$ known.

Jacobi iteration splits a coefficient matrix $A = (D - T)$ into a diagonal matrix D and another matrix T with a zero diagonal. With a modicum of algebra we can derive an iteration $\vec{x}^{(m)} = (D^{-1}T)\vec{x}^{(m-1)} + D^{-1}\vec{b}$ from any problem $A\vec{x} = \vec{b}$. This iteration is known to converge to solutions of the original equation if all eigenvalues of $(D^{-1}T)$ lie within the unit circle in the complex plane. A Jacobi iteration is easy to construct from a given coefficient matrix A. D^{-1} is found by inverting each diagonal term d_i independently, while T is just the negation of A with zero diagonal terms. For a finite difference matrix like A in (6) the resulting $(D^{-1}T)$ is just $d^{-1}T$ where $d = d_i$ is the diagonal term which appears in every row of D. Applying this transformation to the vector equation (6) results in the inner iteration (3).

4. DERIVATION OF PARAMETERS

We have shown that the finite difference representation of the heat equation is accurate to $O(\alpha)$ and we have chosen α to control the quality of the balance which results from simulating the diffusion process. In this section we determine the accuracy of the converged Jacobi iteration and derive a formula for ν which guarantees accuracy α. We then derive a formula for τ which determines the number of "artificial time" steps over which the diffusion occurs.

ACCURACY OF THE JACOBI ITERATION

From the Geršgorin disc theorem [7] we know the eigenvalues λ of (3) are bounded $|\lambda| \leq \frac{6\alpha}{1+6\alpha}$. Since the row and column sums are constant and the iteration matrix is nonnegative we know further ([7], theorem 8.1.22) that the spectral radius equals the row sum

$$\rho\left(D^{-1}T\right) = \frac{6\alpha}{1+6\alpha} \tag{8}$$

Define the error in a current value $\vec{u}^{(m)}$ under the iteration (3) as $e(\vec{u}^{(m)}) = (\vec{u}^{(m)} - \vec{u}^{*})$ where $\vec{u}^{*}$ is the fixed point of the Jacobi iteration. Then for $\nu > 0$

$$e(\vec{u}^{(\nu)}) = e((D^{-1}T)^{\nu}\vec{u}^{(0)}) = (D^{-1}T)^{\nu}e(\vec{u}^{(0)}) \tag{9}$$

which converges when $\rho(D^{-1}T) < 1$ since $\rho((D^{-1}T)^{\nu}) = (\rho(D^{-1}T))^{\nu}$. In order for the algorithm to correctly reduce the error it is necessary to compute the desired load at each time step to an appropriate accuracy. In order to quantify the error define the infinity norm

$$\|e(\vec{u}^{(m)})\|_{\infty} = \max_{x,y,z}\left|e(u^{(m)}_{x,y,z})\right| = \max_{x,y,z}\left|u^{(m)}_{x,y,z} - u^{*}_{x,y,z}\right|$$

Using this norm we can define a necessary condition for improving the accuracy of the solution $\vec{u}$ by a factor α in ν steps to be $\|e(\vec{u}^{(\nu)})\|_{\infty} \leq \alpha\|e(\vec{u}^{(0)})\|_{\infty}$. From (9) we know that this is satisfied when $(\rho(D^{-1}T))^{\nu} \leq \alpha$ and thus (2)

$$\nu = \left\lceil \frac{\ln(\alpha)}{\ln\left(\rho(D^{-1}T)\right)} \right\rceil \tag{10}$$

ELAPSED TIME FOR THE DIFFUSION

In this section we determine the number of artificial time steps τ required to reduce the load imbalance by a factor α. We do this by considering the eigenstructure of the finite difference equation (5) which we rearrange to express the change in load with each iteration

$$\begin{aligned} u_{x,y,z}(t+dt) - u_{x,y,z}(t) = \alpha\,[&u_{x+1,y,z}(t+dt) + u_{x-1,y,z}(t+dt) \\ &+u_{x,y+1,z}(t+dt) + u_{x,y-1,z}(t+dt) \\ &+u_{x,y,z+1}(t+dt) + u_{x,y,z-1}(t+dt) \\ &-6u_{x,y,z}(t+dt)] \end{aligned}$$

or as a vector equation with matrix operator L

$$\vec{u}(t+dt) - \vec{u}(t) = \alpha \mathrm{L}\vec{u}(t+dt) \tag{11}$$

Any load distribution $\vec{u}(t)$ can be written as a weighted superposition of eigenvectors $\vec{x}$ of L

$$\vec{u}(t) = \sum_{i,j,k} a_{i,j,k}(t)\vec{x}_{i,j,k}$$

Using this fact we can rewrite the vector equation (11) as

$$\sum_{i,j,k} a_{i,j,k}(t+dt)\vec{x}_{i,j,k} - \sum_{i,j,k} a_{i,j,k}(t)\vec{x}_{i,j,k} = \alpha \sum_{i,j,k} \mathrm{L}a_{i,j,k}(t+dt)\vec{x}_{i,j,k} \tag{12}$$

Using the definition of $\mathrm{L}\vec{x}_{i,j,k}$ and the eigenvalues of L

$$\mathrm{L}\vec{x}_{i,j,k} = -\lambda_{i,j,k}\vec{x}_{i,j,k}$$

$$\lambda_{i,j,k} = 2\left[3 - \cos\left(2\pi\frac{i}{n^{1/3}}\right) - \cos\left(2\pi\frac{j}{n^{1/3}}\right) - \cos\left(2\pi\frac{k}{n^{1/3}}\right)\right] \tag{13}$$

we can further simplify (12)

$$\sum_{i,j,k} \left(a_{i,j,k}(t+dt)\vec{x}_{i,j,k}\left[1 + \alpha\lambda_{i,j,k}\right] - a_{i,j,k}(t)\vec{x}_{i,j,k}\right) = 0$$

and by the completeness and orthonormality of the eigenvectors

$$a_{i,j,k}(t+dt)\left[1 + \alpha\lambda_{i,j,k}\right] - a_{i,j,k}(t) = 0$$

$$a_{i,j,k}(dt) = \frac{a_{i,j,k}(0)}{1 + \alpha\lambda_{i,j,k}} \tag{14}$$

It is apparent from equation (14) that convergence of the individual eigenmodes is strongly dependent upon the eigenvalues $\lambda_{i,j,k}$. Reducing the amplitude of an arbitrary component i, j, k by α in τ steps of the algorithm requires $[1 + \alpha\lambda_{i,j,k}]^{-\tau} \leq \alpha$. The worst case occurs for the smallest positive eigenvalue $\lambda_{0,0,1} = (2 - 2\cos(2\pi/n^{1/3}))$ which corresponds to a smooth sinusoidal disturbance with period equal to the length of the computational grid. To reduce such a disturbance requires

$$\tau = \left\lceil \frac{\ln \alpha^{-1}}{\ln\left[1 + \alpha\left(2 - 2\cos\frac{2\pi}{n^{1/3}}\right)\right]} \right\rceil \tag{15}$$

Convergence of this slowest component approaches $\ln \alpha^{-1}$ for large n since

$$\lim_{n\to\infty} \ln\left[1 + \alpha\left(2 - 2\cos\frac{2\pi}{n^{1/3}}\right)\right] = 1$$

Convergence of highest wavenumber component $\lambda_{(n^{1/3})/2-1,(n^{1/3})/2-1,(n^{1/3})/2-1}$ is rapid because

$$\tau = \left\lceil \frac{\ln \alpha^{-1}}{\ln\left[1 + (6 - \epsilon)\alpha\right]} \right\rceil \tag{16}$$

These characteristics are well suited to our work in adaptive computational fluid dynamics. The disruptions which occur are localized and can be treated as a series of disruptions at individual processors. For these reasons we consider the time to diffuse an expected case in which in a local area of a large mesh becomes imbalanced. We consider the length of time which must pass before the imbalance is reduced by a factor α. Our conclusion is equation (1).

In the following text we use the Poisson bracket $\langle \cdot, \cdot \rangle$ to represent the inner product operator. When discussing loads or eigenvectors we use $\vec{u}[x, y, z]$ or $\vec{x}_{i,j,k}[x, y, z]$ to denote the vector element which corresponds to location x, y, z of the computational grid with the convention that $[0, 0, 0]$ is the first element of the vector. Then the initial disturbance confined to a particular processor x, y, z can be written as a superposition of eigenvectors of L

$$\vec{u}(0) = \sum_{l,m,n} a_{l,m,n}(0)\vec{x}_{l,m,n} \tag{17}$$

If we assume $u(0)$ to be zero at every element except $[x, y, z]$ then

$$\langle \vec{x}_{i,j,k}, \vec{u}(0) \rangle = \vec{x}_{i,j,k}[x, y, z] \tag{18}$$

This is equal to the initial amplitude $a_{i,j,k}(0)$ of each eigenvector $\vec{x}_{i,j,k}$

$$\begin{aligned} \langle \vec{x}_{i,j,k}, \vec{u}(0) \rangle &= \left\langle \vec{x}_{i,j,k}, \sum_{l,m,n} a_{l,m,n}(0) \vec{x}_{l,m,n} \right\rangle \\ &= \sum_{l,m,n} \langle \vec{x}_{i,j,k}, \vec{x}_{l,m,n} \rangle \, a_{l,m,n}(0) \\ &= \sum_{l,m,n} a_{l,m,n}(0) \delta_{il} \delta_{jm} \delta_{kn} \\ &= a_{i,j,k}(0) \end{aligned} \tag{19}$$

The computational domain has periodic boundary conditions and as a result the origin of the coordinate system is arbitrary. Then without loss of generality we can place the origin at the source of the disturbance and take $x = y = z = 0$. This has no effect on the eigenvectors $\vec{x}_{i,j,k}$ and from (18), (19)

$$a_{i,j,k}(0) = \vec{x}_{i,j,k}[0, 0, 0] \tag{20}$$

Placing the origin at the source of the disturbance is particularly convenient when we consider the first element of the eigenvectors $\vec{x}_{i,j,k}[0, 0, 0]$. L has $\left(n^{1/3}\right)/2$ distinct eigenvalues $\lambda_{i,j,k}$ each of algebraic multiplicity two. Each of these eigenvalues has geometric multiplicity eight, ie. has eight linearly independent associated eigenvectors of unit length

$$\vec{x}_{i,j,k}[x, y, z] = c_{i,j,k} F_1 \left(2\pi \frac{xi}{n^{1/3}} \right) F_2 \left(2\pi \frac{yj}{n^{1/3}} \right) F_3 \left(2\pi \frac{zk}{n^{1/3}} \right) \tag{21}$$

where each F_i is either sin or cos. By choosing $x = y = z = 0$ this expression (21) is zero except for the single eigenvector for which $F_1(x) = F_2(x) = F_3(x) = \cos(x)$. As a result without loss of generality we can restrict our consideration to initial disturbances of the form

$$u[0, 0, 0](0) = \sum_{i,j,k} c_{i,j,k} \vec{x}_{i,j,k}[0, 0, 0] = \sum_{i,j,k} c_{i,j,k}^2 \tag{22}$$

From (14) we define the time dependent disturbance at any location x', y', z'

$$\begin{aligned}\vec{u}[x',y',z'](\tau dt) &= \sum_{i,j,k} c_{i,j,k}\left[1+\alpha\lambda_{i,j,k}\right]^{-\tau}\vec{x}_{i,j,k}[x',y',z'] \\ &= \sum_{i,j,k} c_{i,j,k}^2\left[1+\alpha\lambda_{i,j,k}\right]^{-\tau}\cos\left(2\pi\frac{x'i}{n^{1/3}}\right) \\ &\quad \cos\left(2\pi\frac{y'j}{n^{1/3}}\right)\cos\left(2\pi\frac{z'k}{n^{1/3}}\right) \qquad (23)\end{aligned}$$

In the appendix we show that $c_{i,j,k} = (8/n)^{1/2}$ and thus the disturbance is a summation of equally weighted eigenvectors. ¿From (22) and (23) the time dependent disturbance at $0,0,0$ is therefore

$$\vec{u}[0,0,0](\tau dt) = \frac{8}{n}\sum_{i,j,k}\left[1+\alpha 2\left(3-\cos\frac{2\pi i}{n^{1/3}}-\cos\frac{2\pi j}{n^{1/3}}-\cos\frac{2\pi k}{n^{1/3}}\right)\right]^{-\tau} \qquad (24)$$

Solving $\vec{u}[0,0,0](\tau dt) \leq \alpha$ yields equation (1).

5. ALGORITHM FOR A TWO-DIMENSIONAL MESH

The algorithm for the two dimensional case is very similar to the three dimensional case. The inequality for τ becomes

$$\frac{4}{n}\sum_{i,j}\left[1+\alpha 2\left(2-\cos\frac{2\pi i}{n^{1/2}}-\cos\frac{2\pi j}{n^{1/2}}\right)\right]^{-\tau} \leq \alpha$$

The new spectral radius of the iteration matrix (3) changes ν

$$\nu = \left\lceil \frac{\ln(\alpha)}{\ln\left(\frac{4\alpha}{1+4\alpha}\right)} \right\rceil$$

and similarly the iteration itself

$$u_{x,y}^{(m)} = \left(\frac{\alpha}{1+4\alpha}\right)\left(u_{x+1,y}^{(m-1)} + u_{x-1,y}^{(m-1)} + u_{x,y+1}^{(m-1)} + u_{x,y-1}^{(m-1)}\right) + \frac{u_0}{1+4\alpha}$$

6. CONCLUSIONS

We have demonstrated that by starting with a model of the rebalancing problem and applying established numerical methods to that model we arrive at a scalable and concurrent load balancing algorithm for a mesh-connected multicomputer. The essential features of this algorithm are that it diffuses local disturbances rapidly, it can preserve task locality, and it can rebalance local portions of a domain without interrupting the rest of the computation.

We do not claim that this algorithm is optimal and we identify a worst case in which convergence could be very slow for large n. One strategy to remove this worst case behavior would be to embed this algorithm within a multigrid computation [9] in which balances are computed between coarser subdivisions of the architectural mesh. We are continuing the theoretical work on this problem by considering these sorts of alternative numerical methods for the heat equation model as well as alternative models for load balancing. We are continuing the practical work by implementing this algorithm in an adaptive finite volume flow solver for irregular problems with moving boundary conditions. We expect this experience to give us insight into the policy decisions necessary to develop a practical implementation.

ACKNOWLEDGEMENTS

We are indebted to Andrew Conley for insightful discussions and criticisms.

APPENDIX: NORMALIZATION CONSTANT

In this appendix we demonstrate that the eigenvector normalization constant $c_{i,j,k}$ is equal to $(8/n)^{1/2}$ for all eigenvectors $\vec{x}_{i,j,k}$. ¿From (21) a necessary condition for a normalized eigenvector is

$$\begin{aligned}
1 &= c_{i,j,k}^2 \sum_{x,y,z} \cos^2\left(2\pi\frac{xi}{n^{1/3}}\right)\cos^2\left(2\pi\frac{yj}{n^{1/3}}\right)\cos^2\left(2\pi\frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8}\sum_{x,y,z}\left(1+\cos 4\pi\frac{xi}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8}\sum_{x,y,z}\left\{\left(1+\cos 4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right)\right. \\
&\quad \left. +\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right)\sum_{y,z}\cos\left(4\pi\frac{yj}{n^{1/3}}\right)\left(1+\cos 4\pi\frac{zk}{n^{1/3}}\right)\right\} \qquad (25)
\end{aligned}$$

We can simplify the preceding expression if we make use of the following

Lemma 1

$$\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right) = 0$$

Proof:

$$\begin{aligned}
\sum_x \cos\left(4\pi\frac{xi}{n^{1/3}}\right) &= \sum_x Re\left(e^{i\frac{4\pi xi}{n^{1/3}}}\right) \\
&= Re\sum_x e^{i\frac{4\pi xi}{n^{1/3}}} \\
&= Re\sum_x \left(e^{i\frac{4\pi i}{n^{1/3}}}\right)^x \\
&= Re\left[\frac{e^{i\frac{4\pi i}{n^{1/3}}}\left(1-\left(e^{i\frac{4\pi i}{n^{1/3}}}\right)^{n^{1/3}}\right)}{1-e^{i\frac{4\pi i}{n^{1/3}}}}\right] \\
&= 0
\end{aligned}$$

Q.E.D.

Repeated application of lemma (1) to equation (25) yields

$$\begin{aligned}
1 &= c_{i,j,k}^2 \frac{1}{8} \sum_{x,y,z} \left(1 + \cos 4\pi \frac{yj}{n^{1/3}}\right) \left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right) \\
&= c_{i,j,k}^2 \frac{1}{8} \left[\sum_{x,y,z} \left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right) + \sum_{x,y,z} \left(\cos 4\pi \frac{jy}{n^{1/3}}\right) \left(1 + \cos 4\pi \frac{zk}{n^{1/3}}\right)\right] \\
&= c_{i,j,k}^2 \frac{1}{8} \left[\sum_{x,y,z} 1 + \sum_{x,y,z} \left(\cos 4\pi \frac{zk}{n^{1/3}}\right)\right] \\
&= c_{i,j,k}^2 \frac{1}{8} n \qquad (26)
\end{aligned}$$

From which we conclude

$$c_{i,j,k} = \left(\frac{8}{n}\right)^{1/2} \qquad \forall\, i,j,k$$

References

[1] Anderson, D. A., Tannehill, J. C. & Pletcher, R. H. *Computational Fluid Mechanics and Heat Transfer.* Hemisphere, New York, NY, 1984.

[2] Bertsekas, D. P. & Tsitsiklis, J. N. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[3] Chandy, K. M. & Taylor, S. *An Introduction to Parallel Programming.* Jones & Bartlett, Boston, MA, 1992.

[4] Cybenko, G. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.* **7** (1989), 279–301.

[5] Golub, G. H. & Van Loan, C. F. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1991.

[6] Hong, J., Tan, X. & Chen, M. ¿From local to global: an analysis of nearest neighbor balancing on hypercube. *Proc. 1988 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems.* Association for Computing Machinery, 1988, pp. 73–82.

[7] Horn, R. A. & Johnson, C. R. *Matrix Analysis.* Cambridge University Press, New York, NY, 1991.

[8] Lin, F. C. H. & Keller, R. M. The gradient model load balancing method. *IEEE Trans. Soft. Eng.* **SE-13**, 1 (1987), 32–38.

[9] McCormick, S. F., *Multigrid Methods.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[10] Noakes, M. & Dally, W. J. System design of the J-machine. In Dally, W. J. (Ed.). *Proceedings of the 6th MIT Conference on Advanced Research in VLSI.* MIT Press, Cambridge, MA, 1990, pp. 179–194.

[11] Seitz, C. L. Mosaic C: an experimental fine-grain multicomputer. *Proc. International Conference Celebrating the 25th Anniversary of INRIA, Paris, France, December 1992*, Springer-Verlag, New York, NY, 1992.